AFRL-OSR-VA-TR-2013-0582

SCALABLE AND ACCURATE SMT-BASED MODEL CHECKING OF DATA FLOW SYSTEMS

CESARE TINELLI

UNIVERSITY OF IOWA, THE (INC)

10/30/2013
Final Report

AIR FORCE RESEARCH LABORATORY
AF OFFICE OF SCIENTIFIC RESEARCH (AFOSR)/RSL
ARLINGTON, VIRGINIA 22203
AIR FORCE MATERIEL COMMAND

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
30-09-2013	final	01-06-2009 -- 31-05-2013

4. TITLE AND SUBTITLE

Scalable and Accurate SMT-based Model Checking of Data Flow Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-09-1-0517

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Tinelli, Cesare
Kahsai, Temesghen
Sticksel, Christoph

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The University of Iowa
105 Jessup Hall
Iowa City, IA 52242-1316

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AF Office of Scientific Research
875 N. Ralndolph St. Room 3112
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Publicly Releasable

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Model checking is a form of formal verification that can be used to check certain correctness properties of hardware or software systems mostly automatically. This project's overall objective was to improve the effectiveness and usefulness of model checking for embedded reactive software, the sort of software typically used in avionics. In particular, it sought to extend the scalability of model checking to systems with a very large or infinite state space, improve its accuracy, and develop modular model checking methods for infinite-state systems.
The project achieved its objectives by developing techniques based on automated reasoning engines called SMT solvers, which work with more powerful logics than propositional logic, the logic of choice in traditional model checking. These techniques were implemented in a automated model checker called Kind. Experimental results with Kind show that the new techniques provide superior scalability and precision. The tool has been adopted by several research groups from academia, government and industry. An internal reimplementation of Kind is now used regularly at a major avionics company.

15. SUBJECT TERMS

Model checking, formal verification, reactive software

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Cesare Tinelli
U	U	U	UU		**19b. TELEPHONE NUMBER** *(include area code)* 319-335-0735

Scalable and Accurate SMT-based Model Checking of Data Flow Systems

Grant no: FA9550-09-1-0517

Final Performance Report

1 Introduction

The overall objective of this collaborative project between the University of Iowa and New York University (grant no. FA9550-09-1-0596) was to improve the effectiveness and usefulness of model checking for the debugging and verification of reactive software systems. Specifically, the project aimed at:

- extending the scalability of model checking to systems with a very large or infinite state space;
- developing modular approaches for the model checking of infinite-state systems;
- improving the accuracy of model checking in applications relying on non-linear arithmetic.

The project focused on software systems that could be modeled as synchronous data-flow systems. Its main research hypothesis was that it would be possible to dramatically increase in the scalability and accuracy of model checking for such systems by moving to formulations and automated reasoning engines that rely on more powerful logics than propositional logic. In particular, it considered fragments of first-order logic that can be efficiently decided by solvers for *Satisfiability Modulo Theories* (SMT). The main advantages of using SMT solvers for model checking are that: (i) SMT solvers are quite efficient in practice; (ii) reductions of model checking problems into first-order logic can be exponentially more compact than reductions into propositional logic; and (iii) many infinite-state systems can be modeled directly and faithfully using SMT without resorting to finite-state abstractions. The rationale for moving to SMT-based model checking was that it would enable the development of a new generation of model checkers able to model functional aspects of reactive systems of realistic size and complexity with a high degree of accuracy, and verify their functional properties largely automatically.

The project sought to achieve its three main objectives through the following research components.

RC1: *Develop new SMT-based model checking techniques to verify safety properties of synchronous data-flow systems.*

RC2: *Develop compositional verification methods on top of the SMT-based techniques.*

RC3: *Implement the new techniques in a model checker for systems specified in Lustre.*

RC4: *Develop more accurate methods for automated reasoning in non-linear arithmetic.*

We focus here on Research Components 1, 2 and 3 because they were the main responsibility of the Iowa team. A separate report, for grant no. FA9550-09-1-0596, will be submitted by the NYU team on Component 4 and aspects of the other components that were under its responsibility.

Note. Work on this project at the University of Iowa started about six months after the official start date of July 15, 2009 because of delays in filling the post-doc position funded by it. A corresponding no-cost extension was requested at the time and verbally approved by the then program director, Doctor David Luginbuhl. The PI was, however, instructed to wait until the last year of the project (2013) to make a formal request. This request, made in early 2013, was unexpectedly denied by the current program director. Largely as a consequence of this effective six month cut to the project, some of its specific objectives could not be achieved in full, as explained in the next section.

2 Summary of Significant Work and Results

2.1 Research Component 1

We looked at proving invariant properties of finite and infinite systems with Boolean and numerical variables (both integer-valued and real-valued), initially restricting ourselves to systems devoid of non-linear arithmetic expressions. Such systems can be expressed in an SMT logic with uninterpreted function symbols and linear integer and real arithmetic. A property is invariant for a system if it holds in all of its reachable states. The majority of functional properties of a computation system can be expressed in terms of invariance. As specified in the proposal, we focused our efforts in two main directions, discussed in the following.

2.1.1 Adapt existing induction-based model checking approaches to the new logic

Before the start of the project, the PI and his students had done preliminary work on extending SAT-based k-induction to the SMT logic above. The main feature of that extension is that the system under analysis and any property to check against it encoded as a set of SMT formulas instead of a set of propositional ones. Correspondingly, the problem of checking the invariance of that property is solved by a series of queries to an SMT solver instead of a SAT solver.

A property is *k-inductive* for some $k \geq 0$, if it is satisfied by all system states reachable in up to k steps, and it is preserved by sequences of k-successive states. k-inductiveness is relevant to us because it is a sufficient condition for invariance that can be proven automatically with an SMT solver. k-induction is the process of trying to prove a property k-inductive for some k, usually by starting with $k = 0$ and then trying increasingly large values of k. One main limitation of the k-induction procedure that we used in our previous work is that it was able to check only one property at time. Multiple properties, which occur commonly in realistic cases, can always be checked together as a single conjunctive property P, but this is too course-grained and so not very informative: if some of the properties are invariants and some are not, P itself will not be

invariant. On the other hand, checking one properties at a time can fail, as it is harder for an automated prover based on inductive methods to prove two properties separately than it is to prove them together. Another limitation is that k-induction is geared towards trying to prove properties invariant. If a property is not invariant, other methods such as Bounded Model Checking (BMC) are a lot more effective at showing that.

Enhancements to k-induction We developed a novel parallel, multi-property variant of k-induction to address the two limitation above, together with a parallel architecture for implementing the corresponding model checking procedure. The architecture is strictly message-based and designed to minimize synchronization delays and easily accommodate the concurrent automatic generation of auxiliary invariants to bolster basic k-induction. A first level of parallelism is introduced in the k-induction procedure itself by executing the base step and the inductive step concurrently. The base step is akin to a BMC search and is aimed at disproving that the input properties are invariant by finding executions of the system where the properties fail to hold. The inductive step is aimed at proving the properties invariant by showing that they are k-inductive for some k, starting with $k = 0$ and going increasingly higher. Properties that are disproved by the base step are removed from consideration in the inductive step. Properties that are shown not to be k-inductive by the inductive step for the current k are kept around for the next iteration to see if they are $k + 1$-inductive. Property proved to be k-inductive are used as lemmas to help prove non-k-inductive properties $k + 1$-inductive. A second level of parallelism allows the addition of one or more independent processes that incrementally generate *auxiliary* invariants for the system being verified. These invariants are fed to the k-induction loop as soon as they are produced and used to help prove the input properties by strengthening the k-induction hypothesis.

We implemented this architecture in a parallel model checker for Lustre programs (see Section 2.3). We evaluated this architecture first on benchmarks consisting of a Lustre system and a single property, and then on a smaller but more challenging set of benchmarks consisting of a Lustre system and a several properties. The results of the first evaluation are discussed in [PDMV11] (see Section 3.1); those of the second evaluation are discussed in [NFM12].

2.1.2 Develop new techniques for strengthening induction hypotheses

While k-induction is completely automatable for the SMT logic we considered in this project, it is a fairly weak model checking method. In particular, it might fail to prove large numbers of invariant properties as such (by trying increasingly larger values of k and never succeeding). This is because there exist invariant properties that are not k-inductive for any k. Although with infinite state systems there is no automated procedure that is guaranteed to prove arbitrary invariant properties, it is possible to enhance the basic k-induction process to increase the number of properties it can prove. An effective way to do that is to enrich the description of the system with the assertion of *other* properties that are already known to be invariant for that system. Strengthening the system's description this way can turn a previously non-k-inductive property into a k-inductive one (increasing k-induction's precision), or make a k-inductive property k'-inductive for a considerable smaller k' (increasing speed). Since auxiliary invariants are notoriously difficult or time-consuming to specify manually, an important research problem is how to generate them automatically.

Brute-force invariant generation In much of previous work, invariants are synthesized from the system's description, using sophisticated algorithms guided by the semantics of the description language. In this project we developed instead a complementary and novel approach based on a somewhat brute-force invariant discovery scheme—which nonetheless proved to be effective in practice. The approach looks for possible invariants by sifting through a large set of automatically generated formulas expressing state properties. These formulas are all instances of the same template, the parameter of the scheme, representing a decidable relation over one of the system's data types. Specifically, we devised a general invariant discovery scheme that, a template formula $R[x, y]$ in two arguments, looks for invariants that are instances $R[s, t]$ of the template where s and t are terms chosen from a some set U of *interesting* terms. The set U can be constructed heuristically in any number of ways from $\mathcal{S}$ and a given set of properties to be proven invariant for $\mathcal{S}$.

The general scheme relies on the availability of efficient reasoning engines, such SMT solvers, and capitalizes on their ability to quickly generate counter-models. It consists of a two-phase procedure, with an optional third phase not discussed here. Given the template R and the term set U, the first phase starts with the (very crude) conjecture that the state property $C = \bigwedge_{s,t \in U} R[s, t]$ is invariant. Then, it uses the SMT solver to weaken that conjecture by eliminating from it as many conjuncts $R[s, t]$ as possible that have a counterexample—specifically, all conjuncts falsified by a k-reachable state, for some heuristically determined k. The resulting formula C is passed to the second phase, which attempts to prove C k-inductive by checking that it satisfies the inductive step of k-induction. Any counter-examples there are used, conservatively, to weaken C further by eliminating additional conjuncts until no counter-examples exists. The final formula—the empty conjunction in the worst case—is by construction k-inductive, and so invariant.

The scheme above is impractical in its full generality because the number of instances of R over U can be very large. So we devised two specializations to relations R that are partial orders, one for general posets and one specific to binary posets. These specializations rely on the properties of partial orders to represent the conjunctive conjecture C compactly, and weaken it efficiently.

We devised an incremental, on-line version of the procedure above that generates k-inductive invariants progressively, for successive values of k. As templates R we used the $\leq$ relation over integer terms and the $\Rightarrow$ (logical implication) relation over Boolean terms. In the context of the parallel model checking architecture described above, the auxiliary invariants of the form $s \leq t$ or $s \Rightarrow t$ are generated concurrently to the base and inductive steps of k-induction, and are sent to the latter step as soon as they are generated. This approach was rather successful experimentally, yielding considerably improvements in terms of the number of property that k-induction can prove. For instance, focusing on single property benchmarks, auxiliary invariant generation was able to improve the precision of our k-induction procedure from 61% to 85% on a set of about 500 benchmarks of different size, complexity and origin. More details on our invariant generation procedure and on our experimental results can be found in [NFM11].

Mode invariant generation We developed a further enhancement to our invariant generation procedure based on the realization that embedded systems often contain complex modal behavior that describes how the system interacts with its environment. In these systems, the modes of the software drive the behavior of the device. For instance, in a flight guidance system, an approach mode enables a control procedure that attempts to land the airplane, while a go-around mode enables a controller that attempts to climb the aircraft to a suitable safe altitude. These modes are often designed as state machines or mode transition tables. In addition, embedded

systems typically have several parallel mode machines that communicate with one another to define the control state of the system. Understanding which variables in a systems model are *mode variables*, i.e., represent system modes, and discovering relationships between such variables often determines whether a safety property can be proven or not.

We developed a method for identifying likely mode variables automatically, and discovering invariant relationships among them. We did that by adapting the template-based invariant generation method described above. The new method heuristically considers as a mode variable any system variable that (i) ranges over a (small) finite set of values and (ii) whose next-state value is determined in part by its current value. We generalized this idea slightly to *mode variable sets* in which strongly-connected variables define a particular system mode. Then we developed a general invariant generation method to identify implicative relationships between values of mode variables (e.g., $x = 4 \Rightarrow y = 1$).

We implemented the method in our *k*-induction model checker and evaluated it on a complex hierarchical problem, made publicly available by NASA. This is the formal model of a system that controls the approach behavior of the Space Shuttle when docking with the International Space Station. As the shuttle approaches the ISS it goes through several operational modes related to how the shuttle is to orient itself for capture, dock with the ISS, and capture the ISS docking latch, among several other operational modes. The model describing this behavior is quite intricate and consists of a hierarchical and parallel state machine with three levels of hierarchy and multiple parallel state machines, including a total of 64 states. In this model, there are several properties related to invariants that must be true in certain stages of approach, and sequencing conditions that describe allowable paths through the state machine. The model is quite difficult to analyze and it had previous required hand-written mode invariants to prove interesting properties using k-induction solvers. For the purposes of this experiment, we created five reduced versions of the docking approach model in which we replaced one of the complex hierarchical states with a simple state that approximates its behavior. This allowed us to examine the behavior of our invariant generation method over a range of state machine models with different characteristics (the hierarchical states vary substantially in size).

We ran our model checker in various configurations. The basic one, *k*-induction with no invariants, was able to prove only 42% of all properties over the 5 models. Activating the invariant generation method described earlier improved precision to %80. Adding mode invariant generation allowed our checker to push that up further to %91. Details on this work and the experimental evaluation be found in [NFM12].

Invariant generation with SMT-based automatic abstract transformers Abstract interpretation is currently one of the best approaches for discovering useful invariants for a system, in particular numerical ones. However, its application is limited by two orthogonal issues: (i) developing an abstract interpretation is often non-trivial; each transfer function of the system has to be represented at the abstract level, depending on the abstract domain used; (ii) with precise but costly abstract domains, the information computed by the abstract interpreter can be used only once a post fix point has been reached; this may take a long time for large systems or when widening is delayed to improve precision. We developed a new, completely automatic, method to build abstract interpreters which, in addition, can provide auxiliary invariants for the system under analysis *before* reaching the end of the post fix point computation. In effect, such interpreters act as on-the-fly invariant generators and can be incorporated in the parallel architecture described

earlier, again to improve the precision of the main k-induction component.

While motivated by practical issues (namely, the generation of invariants for k-induction model checkers) our method is more general and can be adapted to a wide variety of contexts. It only requires that a system's semantics be expressible in a decidable logic with an efficient solver, such as SAT or SMT solvers, and that the elements of the chosen abstract domain be effectively representable by formulas in that logic. Such requirements are satisfied by a large number of abstract domains used in current practice. As a consequence, we believe that our approach could help considerably in expanding the reach of abstract interpretation techniques to a variety of target languages, as well as facilitate their integration with complementary techniques from model checking.

Since our original motivation was proving invariant properties of Lustre programs expressible in our SMT logic we built an abstract interpreter, called Kind-AI, for such Lustre programs that computes the abstract transformer automatically, and generates a stream of invariants during its fix point computation. As abstract domain we use one defined, as usual, as a reduced product of a variety of abstract numerical domains, including relational and non-relational ones. Our experimental results were encouraging although not as strong as for the other methods. On the other hand, our implementation was relatively unsophisticated as far as abstract interpretation tools go, so it is plausible that further work on that would produce more impressive result.

Details of our abstract interpreter construction and our experimental results can be found in [NSF'13].

2.2 Research Component 2

Work on this component was cut short by the early termination of the project. We worked on developing a framework for compositional verification that capitalizes on the fact that data-flow languages, such as Lustre, have direct support for modularization, module reuse, and composition. Systems described in these languages not only tend to be highly modular but also often come equipped with a specification of local invariants for each sub-module. We designed our framework to take advantage of the preexisting decomposition of the overall system and the environmental assumptions already provided by its developers.

We started to develop an abstraction and refinement mechanism that abstract submodules by their (separately proved) invariants. These invariants could be either directly provided by the user or computed by the model checker itself using the invariant generation methods developed in Research Component 1. The advantage of this approach would show in systems with large or complex submodules in cases when, to prove a property of the top-level system, it is enough to know only a few invariant properties of the visible behavior of the submodule, as opposed to its full definition. In those cases, scalability can be improved by replacing the module by its properties during the model checking process. Unfortunately, we did not have enough time to complete the development and the implementation of the framework even if we have much of the low-level infrastructure in place. As a consequence, we do not have any experimental evaluation to report for this research component.

2.3 Research Component 3

We implemented all the methods described above in the Kind model checker, a model checker written in OCaml that we had started to implement before this project. A major expansion and

refactoring of the checker was done in mid project to turn Kind into a parallel model checker, called PKind, with support for multiple properties and incremental invariant generation. Both PKind and Kind are publicly available in source form and have a very liberal free-software license.

Last year it became clear that PKind, although quite successful, was not the best platform to continue our investigations, for a number of reasons: it was burdened by legacy code, a lot of which corresponded to several variants of our techniques that we had produced for experimentation purposes; it had not been written from the beginning to be parallel and so was suboptimal in some respects; it was unnecessarily tied to Lustre, whereas our techniques were general enough to be applicable to system models expressed in a variety of languages. Hence in the summer of 2012 we decided that it was time to start from scratch implementation-wise. Since then we have focused on the development of the next generation of the model checker, called Kind 2, while doing minimal development and maintenance of PKind and its predecessor—both of which we will collectively refer to as Kind 1 in the following.

Kind 2 is a from-scratch reimplementation of Kind 1 that builds on the experience and expertise we accumulated during the development of Kind 1. It starts with a much cleaner design and architecture distilling the features that have proven most useful and effective in Kind 1, and provides the basis for further research and development. Kind 2 was designed from the beginning on to be parallel based on a lightweight message-passing architecture, extensible to support multiple novel model checking engines, and easily portable to different operating systems. Also, it provides a clean and complete separation between the front end and the model checker proper through a client-server architecture. It is designed to support a variety of front ends, both local and remote, and more than one model specification language. Support for Lustre and a textual interface, as in Kind 1, is maintained to facilitate the migration from Kind 1.

In terms of its internal architecture, Kind 2 still has basic components implementing multi-property k-induction with auxiliary invariant generation, similar to Kind 1. However, it adds other model checking methods, starting with a property directed reachability (PDR) method which extends to infinite-state systems. This method, originally called IC3 and developed by Aaron Bradley at the University of Colorado, has shown to be highly successful in the area of hardware model checking. In Kind 2, components implementing this method work in parallel with those on k-induction and cooperate with them through the exchange of invariants and information on proved and disproved properties. To enable the exchange of invariants, Kind 2 provides a general invariant collection and dispatching mechanism, connecting multiple invariant producers to multiple consumers (with certain components acting in both roles).

Our expectation is that, thanks to its fresh implementation and innovative architecture, Kind 2 will be a better platform for further research while at the same time serving the needs of the core Kind 1 user base, which includes users at Universities (Iowa, Minnesota), governmental agencies (NASA, Onera) and companies (Rockwell Collins).

Kind 2 is currently available only as an alpha release with limited features, although those features are fully functional. Kind 2 is built, again in OCaml, on a low level infrastructure that allows it to have a client-server architecture where the system's front end is a client and the model checker proper is a server that can be located on a different network node. This will facilitate the creation of different front ends (interactive/batch, text-based/GUI-based/web application) and allow the client and the server to run even on nodes with different operating systems.

The development of a PRD engine of Kind 2 was motivated by the success of the original IC3 procedure and the question of whether this success could be replicated with infinite-state systems.

Table 1: Precision of Kind 1 and Kind 2 on 925 single-property benchmark problems with both valid and invalid properties

(a) Number of problems proved

	Properties	
	Valid	Invalid
(1) Kind 1	292	438
(2) Kind 2, *k*-ind.	258	384
(3) Kind 2, PDR	292	345
(4) Kind 2, *k*-ind. + PDR	331	389

(b) Problems proved exclusively by one version

	Properties	
	Valid	Invalid
(1) only	55	93
(3) only	55	0
(1) and (4)	237	345

Like *k*-induction, IC3 uses inductive arguments to prove that a given property P is invariant for a given system. However, it works rather differently. One of its crucial aspects is the generalization of counterexamples to its induction steps, from single states to entires sets of them. Our PDR procedure is an SMT-based extension of IC3 whose distinctive feature is a general method for the generalization of induction counterexamples based on an efficient form of approximate quantifier elimination. The approximation is computed in two stages and is driven by counterexample. An initial experimental evaluation indicates that our approach makes quantified elimination practical. In particular, it is competitive with advanced *k*-induction implementation in Kind 1. Interestingly, the two approaches show complementary strengths, suggesting that a parallel combination of the two can be beneficial.

We have completed the implementation of the PDR engine for Kind 2, and added an initial implementation of a *k*-induction engine similar to the one in Kind 1. Initial comparisons of the performance of Kind 2, using the PDR engine only, with Kind 1, using *k*-induction show that for valid properties, Kind 2's precision is already competitive with that of Kind 1. Interestingly, each system can prove valid properties that the other cannot prove. Moreover, they show different runtimes on the majority of the properties they can both prove. Therefore, we expect that the PDR engine and the reimplementation of the *k*-induction engine of Kind 1 will complement each other and considerably improve the performance of the overall system, both in terms of precision and runtime.

Both engines run in parallel, where the *k*-induction part itself is composed of two separate parallel processes for the base and the induction step. All three processes, as well as future invariant generators and model checking engines, are controlled by and communicate through a central invariant collector and dispatcher. We evaluated Kind 1 (PKind) against Kind 2 with combinations of the *k*-induction and PDR engines enabled and disabled. The results were obtained on

large compute cluster at the University of Iowa. Since this cluster is a shared resource and complete isolation from other jobs on a compute node was not always possible, the runtime results are not completely accurate, but even accounting for this noise in the datasets, there are certainly significant trends.

Table 1 shows the precision of Kind 1 and Kind 2 on a set of 925 single-property benchmark problems that contains both invariant and non-invariant properties. Kind 2 with only the *k*-induction engine enabled lags behind Kind 1, which only has a *k*-induction engine. This was expected because Kind 2's *k*-inductive engine is still fairly basic, lacking several high- and low-level enhancements present int Kind 1. That said, Kind 2 with only the PDR engine shows comparable performance; more importantly, proves a number of properties that Kind 1 cannot solve. The combination of the *k*-induction engine with the PDR engine is superior to Kind 1 in the number of proved invariant properties. Since in either system non-invariant properties are mostly discovered by the bounded model checking component of *k*-induction, Kind 1 has an advantage here. However, there are a few properties disproved only with the PDR engine.

More details on our extension of PDR to the infinite-state case using SMT solvers and on its implementation in Kind 2 can be found in a technical report submitted for conference publication and available at `http://clc.cs.uiowa.edu/Kind/TACAS14/`.

3 Significant Accomplishments

3.1 Archival Publications

Note Extended versions of the publications below, together with detailed experimental results, are available at `http://clc.cs.uiowa.edu/Kind`.

[NFM11] T. Kahsai and Y. Ge, and C. Tinelli. Instantiation-Based Invariant Discovery. In M. Bobaru, K. Havelund, G. Holzmann, R. Joshi editors, *Proceedings of the 3rd NASA Formal Methods Symposium*. Volume 6617 of *Lecture Notes in Computer Science*. Springer, 2011.

[PDMV11] T. Kahsai and C. Tinelli. PKind: A parallel k-induction based model checker. In J. Barnat and K. Heljanko editors, *Proceedings of the 10th International Workshop on Parallel and Distributed Methods in Verification*. Volume 72 of *Electronic Proceedings in Theoretical Computer Science*. 2011.

[NFM12] P.-L. Garoche, T.Kahsai, C. Tinelli and M. Whalen. Incremental verification with mode variable invariants in state machines. In A. Goodloe and S. Person editors, *Proceedings of 4th NASA Formal Methods Symposium*. Volume 7226 of *Lecture Notes in Computer Science*. Springer, 2012.

[NFM12b] C. Tinelli. SMT-Based Model Checking. In A. Goodloe and S. Person editors, *Proceedings of 4th NASA Formal Methods Symposium*. Volume 7226 of *Lecture Notes in Computer Science*. Springer, 2012.

[NFM13] P.-L. Garoche, T. Kahsai and C. Tinelli. Incremental Invariant Generation using Logic-based Automatic Abstract Transformers. In G. Brat, N. Rungta and A. Venet editors, *Proceedings of the 5th NASA Formal Methods Symposium*. Volume 7871 of *Lecture Notes in Computer Science*. Springer, 2013.

Invited talks

1. *SMT-based model checking and verification of data flow programs*. University of Delaware, Newark, DE. October 2009.
2. *Scalable and Accurate SMT-Based Model Checking of Data Flow Systems*. Safe and Secure Systems and Software Symposium. June 2010.
3. *SMT-based model checking*. Summer School on Formal Techniques, Atherton, CA. May 2011.
4. *Incremental and parallel model checking for synchronous systems*. Safe and Secure Systems and Software Symposium. June 2011.
5. *Incremental SMT-based model checking of synchronous systems*. Microsoft Research, Redmond, WA. April 2012.
6. *SMT-based model checking*. NASA Formal Methods Symposium, Norfolk, Virginia, USA. April 2012.
7. *Incremental SMT-based model checking of synchronous systems*. United Technologies Research Center, October 2012.
8. *Incremental SMT-based model checking of synchronous systems*. Max Planck Institute for Computer Science, Germany. July 2012.
9. *SMT-based safety checking*. The University of North Carolina, Charlotte. April 2013.

Software

- The Kind and PKind model checkers. Model checkers for Lustre programs mainly developed in this project. Available in source format at `http://clc.cs.uiowa.edu/Kind`.
- The Kind 2 model checker. From scratch-reimplementation of (P)Kind. Currently available in an alpha-release version at `https://github.com/kind-mc`.

Impact

- Various version of the Kind model checker have being used extensively by researchers at Rockwell-Collins; the University of Minnesota; NASA, Ames; Verimag, a French research center in embedded system; ONERA, a French aeronautics, space and defense research lab.
- While we did not collect statistics on the number of downloads of Kind, its website has had about 4,000 unique visits since December 2010 from around the world; of these, about 2,700 where first time visits. This indicates a strong wide interest in the tools produced by this project.
- The most substantial external user of Kind is Rockwell-Collins which has also implemented from scratch an internal version closely based on PKind but leaner and written entirely in Java—for better integration in they tool chain.[1]

[1] `https://github.com/agacek/jkind`.

- The initial results obtained in this project inspired a collaboration with ONERA which resulted in a year long visit to Iowa by one of their researchers, Pierre-Loïc Garoche.
- The work done on this project has also led to two collaborative projects on related topics with Rockwell Collins and the University of Minnesota, one funded by AFRL and one by NASA.
- It has also enabled an initial exploratory collaboration with United Technologies Research center, funded by them, on possible applications of Kind to their verification problems.
- As a consequence of the expertise on model checking of data flow language that the PI acquired in this project, he was invited as keynote speaker at the 2010 NASA Formal Methods Symposium (the other two keynote speakers were Patrick Cousout and Andrew Appel).
- For the same reason, he was also invited by the prestigious École Normale Supérieure in Paris to visit there for a month in 2014 to work with ENS faculty on further applications of the model checking technology developed on this project.